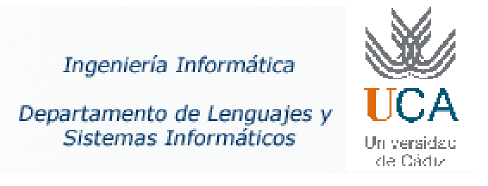




REPRESENTACIÓN DEL CONOCIMIENTO – CLOS (CLIPS)

Práctica 3

Objetivos: Implementación de Marcos mediante las características de CLOS en CLIPS. Definición y manejo de clases e instancias.

1. DEFINICIÓN DE CLASES

La clase raíz en CLIPS se denomina: **OBJECT,** y **USER** es una subclase de object predefinida en CLIPS. Para definir clases se usa **defclass.** Por convención los nombres de las clases se escribirán en mayúsculas.
Es muy recomendable que las clases que se definan sean subclases de USER, ya que así pueden heredar las funciones print, init y delete.

```
(defclass <NOMBRE-CLASE> (is-a <superclases directas>)
([role concrete| abstract])
([slot/multislot  <nombre-campo> <facetas>]))
```

role concrete: permite que se puedan crear instancias de esta clase, POR DEFECTO se asume este role
role abstract: NO permite instancias directas

**** Si una clase es subclase de una abstracta heredará este role a menos que se indique lo contrario con role concrete*

EJEMPLO 1

Todos los romanizados hablan latín

```
(defclass ROMANIZADOS (is-a USER)
(role abstract)
(slot habla (default latin)))

(list-defclasses)
```

Todos los galos han sido romanizados. A todos los galos les gusta el jabalí.

```
(defclass GALOS (is-a ROMANIZADOS)
(role concrete)
(slot comida_favorita (default jabali)))
(browse-classes)
```

Los habitantes de Lutecia son galos.

```
(defclass LUTECIANOS (is-a GALOS))
(browse-classes ROMANIZADOS)
```

Funciones sobre las clases

```
(superclassp <clase1> <clase2>)  devuelve TRUE si clase1 es superclase de clase2
(subclassp <clase1> <clase2>)
(list-defclasses)
(browse-classes  [<clase>])   se puede especificar el nombre de la clase a partir de
donde empezar a mostrar la jerarquía de clases
(ppdefclass <clase1>)
(undefclass <clase1>) Elimina una clase
(describe-class <clase1>)
(class-abstractp <clase1>)  Devuelve TRUE si la clase es abstracta
```

2. INSTANCIAS

```
(make-instance  [<nombre  instancia>]  of  <clase>  <redefinición  de
slots>]
```

Funciones sobre instancias

```
(instances)
(unmake instance <[instancia]>),  con el * se borran todas
(reset) borra todas las instancias creadas y deja sólo initial-instance (como con los hechos)
(symbol-to-instance-name …) convierte un símbolo en una nombre de instancia
(instance-name-to-symbol ..) convierte un nombre de instancia en un símbolo
```

Fransuá es un habitante de Lutecia.

```
(make-instance Fransua of LUTECIANOS)
(instances)
(send [Fransua] print)
```

Los habitantes del poblado de Astérix (como el propio Astérix) son galos pero no han sido romanizados ...

```
(defclass REBELDES (is-a GALOS)
 (slot habla (default lengua-no-latina)))

(make-instance Asterix of REBELDES)
(send [Asterix] print)
```

... salvo Infiltrádix que, a pesar de vivir en el poblado de Astérix, se puede considerar completamente romanizado.

```
*** crea la instancia correspondiente a Infiltrádix

(makde-instance Infiltradix of .........
```

Al igual que ocurre con los hechos, las instancias pueden definirse con **definstances**, cuando una orden **reset** se ejecuta, las instancias así definidas serán recargadas automáticamente.

```
(definstances  amigosAsterix “Amigos de Asterix”
      (Asterix of ALDEANOS_POBLADO_ASTERIX)
      (Obelix of ALDEANOS_POBLADO_ASTERIX)
      (Fransua of LUTECIANOS)
      (Infiltradix of ALDEANOS_POBLADO_ASTERIX
                   (habla latin)))

(instances)
```

⇒ Hasta que no se ejecute la orden reset, no aparecerán estas instancias definidas

```
(reset)
(instances)
```

3. SEND

La función SEND es la que permite gestionar los distintos objetos de las clases. El formato general es (el nombre de la instancia tiene que ir entre corchetes):

```
(send  [<instancia>]  <mensaje>)
```

Ejemplos:

```
(send [Infiltradix] print)

(send [Obelix] print)

(send [Fransua] print)
```

get / put

Todos los slots por defecto poseen dos funciones asociadas a la instancia: **get** y **put**, que sirven para obtener/asignar un valor a un slot. (También se obtienen cuando se utiliza para un slot la faceta **create-accessor**)
Veamos con un ejemplo:

EJEMPLO 2

```
(defclass  PERSONA (is-a USER) (role concrete)
      (multislot nombre)
      (slot  dni)
      (slot  edad (create-accessor read-write)))

(definstances  OBJETOS_PERSONAS
      (Manuel of PERSONA (nombre Manuel Rivas Maldo))
      (Maria of PERSONA  (nombre Maria Castelar Cas)))

(reset)
(send [Manuel] put-dni 3334444)
(send [Manuel] put-edad 23)

(send [Manuel] print)
```

```
(send [Manuel]  get-dni)
(send [Manuel]  get-edad)

(send [Maria]  get-edad)
```

4. FACETAS

Las facetas definen las características de los slots.

default **default-dynamic**	<valores>	Valores iniciales de los slots (campos), pueden aparecer expresiones pero que no contengan variables. Con default-dynamic cada vez que se crea una instancia se re-evalúa la expresión.
cardinality	<numero>	Número de valores
access type	**read-write** **read-only** **initialize-only**	Tipo de acceso, por defecto de lectura y escritura. Read-only el valor del slot se establece con la faceta default (la anterior). Initialize-only como read-only, el valor se le puede dar en make-instance y con handlers de tipo **init.**
storage	**local** **shared**	Valor local a una instancia concreta shared (clases): si se modifica en una instancia se modifica en todas las demás instancias
propagation	**Inherit** **no-inherit**	Para indicar si las clases indirectas pueden heredar o no el slot que tiene esta faceta.
create-accessor	**read-write** **read** **write** **?NONE**	Para cada slot se crea get y put Para cada slot se crea sólo get Para cada slot se crea sólo put No crea ninguna función de acceso al slot
visibility	**prívate** **public**	Clases públicas o privadas
pattern-match	**reactive** **non-reactive**	Permitir que puedan equipararse en las reglas (por defecto reactives)

(consulta un manual de usuario si deseas completar la tabla)

La faceta **storage** define el lugar donde se almacena el valor de un slot: en la instancia (local) o en la clase (shared). Si se almacena local en la instancia, sólo la instancia concreta conoce su valor. Si se almacena en la clase con **storage shared** el valor es el mismo para todas las instancias y si se modifica en un slot, se modificará en todas las instancias.

Redefine la clase PERSONA, añadiendo un nuevo campo para almacenar la nacionalidad:

1. especificando la faceta **storage shared**
2. para la instancia Manuel, asigna nacionalidad gala, y verifica qué nacionalidad tiene María (sin haber especificado ningún valor previo)
3. asigna a María nacionalidad espagnola y verifica qué pasa con la nacionalidad de Manuel

pattern-match permite que un slot pueda activar una regla o no:

EJEMPLO 3

```
(defclass ANIMAL (is-a USER)     (role concrete))

(defclass GATO (is-a ANIMAL)
   (slot peso (create-accessor write) ))
```

```
(defclass PERRO (is-a USER)
   (slot peso (create-accessor write) (pattern-match non-reactive)))

(defrule crear
   ?instancia <- (object (is-a GATO | PERRO))
   =>
   (printout t "La instancia creada es " crlf)
   (send (instance-name ?instancia) print)
   (printout t crlf crlf ))


(defrule acceso-peso
   ?instancia <- (object (is-a GATO| PERRO) (peso ?p))
   =>
   (printout t "peso: " ?p  crlf))

(make-instance Silvestre of GATO (peso 5))
(make-instance Pluto of PERRO (peso 8))

(agenda)

(run)

(send [Silvestre] get-peso)
```

DEFINICIÓN DE GESTORES DE CLASES (HANDLERS)

(defmessage-handler <CLASE> <nombre función> [<tipo handler>] [<argumentos>]
(<lista de acciones>)**)**

Al igual que con las funciones, sólo se devuelve el valor calculado en la última acción.

Existen cuatro categorías de gestores:
primary: realiza la mayor parte de la tarea para un mensaje.

before: realiza las acciones previas a que un primary se ejecute.
after: realiza las acciones posteriores a que un primary se ejecute.
around: establece un entorno desde el cual poder ejecutar el resto de gestores. Comienza antes que ningún otro handler y finaliza después que ningún otro.

- Por defecto los gestores son primary.

?self : es una variable especial para almacenar el nombre de la instancia actual, y se puede utilizar en las acciones pero no como argumento

Para leer el valor de un slot:

?self:<slot>

EJEMPLO 2

(A partir de la clase PERSONA)

Gestor para imprimir la edad de una persona en pantalla:

```
(defmessage-handler PERSONA  imprimir-edad ()
      (printout t "**** imprimiendo ....   *****"  crlf
                  "     EDAD = " ?self:edad crlf
                  "**** fin de la impresión *****" crlf))
)
(send [Manuel] imprimir-edad)
```

A veces es conveniente utilizar **dynamic-get** y **dynamic-put,** ya que de esta forma se hace referencia al valor más actual, aunque haya sido refedefinido en una instancia concreta.

```
(defmessage-handler PERSONA quitando-edad (?agnos-menos)
            (bind ?nueva  (- ?self:edad  ?agnos-menos))
            (dynamic-put edad ?nueva)
            (printout t "**** Rejuveneciendo a " ?self:nombre crlf
                  " ahora tiene " ?nueva "   ******" crlf )
)

(send [Manuel] quitando-edad 3)

(send [Manuel] print)
```

Gestores predefinidos de tipo Primary.
*(con * los que pueden ser llamados directamente por el usuario con* **send [**instancia**]** handler)

init	El usuario no lo utiliza directamente, se llama a través de make-instance o initialize-instance.
*** delete**	Borrar una instancia, devuelve el valor TRUE cuando se borra con éxito **send** [instancia] **delete**
*** print**	Mostrar una instancia y sus campos en pantalla
direct-modify	Para modificar una instancia directamente en vez de utilizar put. Lo utilizan las funciones modify-instance y active-modify-instance.
message-modify	Modifica los slots de una instancia usando put para cada slot. Las funciones **message-modify-instance** y **active-modify-instance** lo utilizan.
direct-duplicate	Duplica una instancia sin usar put, lo usan **duplicate-instance** y **active-duplicate-instance.**
message-duplicate	Duplicar una instancia usando put y get, lo usan **message-duplicate-instance** y **active-message-instance.**
create	Se llama después de que se cree una instancia y antes de que se inicialicen los slots.

DEMONIOS (DAEMONS)

Son procedimientos invisibles que se ejecutan automáticamente siempre que tiene lugar una acción básica tal como inicializar, acceder, borrar o modificar el valor de un campo.
Estas acciones básicas son gestores de tipo primary (normalmente predefinidas), los daemons son definidos por el usuario y de tipo **after, before o around**.

EJEMPLO 3

En este ejemplo a partir del handler predefinido para borrar una instancia: **delete**, se implementarán dos daemons, uno que se ejecuta antes y otro que se ejecuta después de hacer la llamada con **send**. No llevan ningún argumento, por eso los paréntesis están vacíos.

```
(clear)
(defclass ANIMAL (is-a USER) )

(defmessage-handler ANIMAL  delete   before  ()
      (printout t  “   En proceso de eliminación de una instancia de
ANIMALES .... “ crlf ))

(defmessage-handler ANIMAL  delete   after  ()
      (printout t  “   La instancia ha sido completamente eliminada
.... “ crlf ))

(make-instance Fido of ANIMAL)
(instances)

(send [Fido] delete)
```

Observa la ejecución de esta orden qué produce.

EJEMPLO 4

En este ejemplo se crea un daemon para que antes de asignar un valor al slot x (con put-x) de la clase A, muestre un mensaje avisando del valor que va a tomar.

```
(clear)
(defclass A (is-a USER)
      (slot x (default 34)(create-accessor write))
      (slot y (default abc)))

(defmessage-handler A put-x before (?value)
      (printout t " El Slot x va a tomar el valor " crlf))

(make-instance a  of A)
(send [a] put-x  55)
```

¿qué ocurrirá en este caso?

```
(make-instance aa of A (x 22))
```

Ejemplos con MULTISLOTS

Los slots pueden contener un único valor o varios (**multislot).**

CREACIÓN

(create$ <expresion>)
Devuelve un valor multislot (lista de valores) :

```
(create$ Gato Perro Ratón)
(create$ (+ 1 2) (* 2 3))
```

ACCESO

Las funciones **nth$** y **legnth$** se pueden utilizar para acceder a valores concretos dentro de la lista de un multislot.

```
(defclass PERSONA (is-a USER)
      (multislot nombreyapellidos)
      (slot dni)
      (slot edad) )
(make-instance Manuela of PERSONA
      (nombreyapellidos Manuel Barbera Cantero ) (dni 33377999) (edad 28))

(nth$ 2 (send [Manuela] get-nombreyapellidos))
```